Low-level feature extraction

Feature point detection

Image Feature Extraction Techniques

Piotr Fulmański

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
University of Lodz

Difference of Gaussians (DoG)

Gausian averaging operator

From *Basic image processing. Group operators* lecture we know *Gaussian averaging operator* which takes the form:

$$g(x, y, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2 + y^2}{2\sigma^2}}.$$

We use it to blur image. The blurred image is obtained by convolving the original grayscale image with Gaussian kernels (calculated with the following formula for a given window size and deviation σ).

Difference of Gaussians (DoG)

Algorithm

Difference of Gaussian is super simple.

1. Choose two different Gaussian kernels selecting two different variances σ_1 and σ_2, and one common window size (if you want, you can choose two windows of different size).

2. Convolve twice the image with those Gaussian kernels separately — you will obtain two differently blurred images.

3. Subtract one image from another.

4. Threshold image from previous step to filter out the pixels with weaker intensity.

Difference of Gaussians (DoG)

Algorithm

If

$$g_{\sigma_1}(x, y) = g(x, y, \sigma_1) = \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-\frac{x^2 + y^2}{2\sigma_1^2}}$$

$$g_{\sigma_2}(x, y) = g(x, y, \sigma_2) = \frac{1}{\sqrt{2\pi\sigma_2^2}} e^{-\frac{x^2 + y^2}{2\sigma_2^2}}$$

$$g_1(x, y) = g_{\sigma_1}(x, y) * I$$

$$g_2(x, y) = g_{\sigma_2}(x, y) * I$$

$$DoG(x, y) = g_{\sigma_1}(x, y) - g_{\sigma_2}(x, y)$$

$$g_1(x, y) - g_2(x, y) = DoG(x, y) * I$$

Difference of Gaussians (DoG)

Remarks

The difference of Gaussians can be utilized to increase the visibility of edges and other detail present in a digital image.

By blurring you remove components that represent noise (they are known as high-frequency components), and by subtracting you remove some components that correspond to the homogeneous areas in the image (they are known as low-frequency components).

Thus, the Difference of Gaussian acts like a bandpass filter.

The remaining components (frequency components) are assumed to be associated with the edges in the images.

Approximate Laplacian of the Gaussian (LoG) with DoG

Blobs

In computer vision blob is a region in a digital image that differ in properties, such as brightness or color, compared to surrounding regions. Informally, a blob is a region of an image in which some properties are constant or approximately constant; all the points in a blob can be considered in some sense to be similar to each other.

Laplacian of the Gaussian (LoG)

Remarks

One of the first and also most common blob detectors is based on the Laplacian of the Gaussian (LoG). Given an input image $f(x, y)$, this image is convolved by a Gaussian kernel:

$$g(x, y, \sigma) = \frac{1}{2\pi\sigma} e^{-\frac{x^2 + y^2}{2\sigma}}$$

at a certain scale σ (blur ratio) to give a scale space representation (see [2] for examples of different scale applied to an image):

$$L(x, y; \sigma) = g(x, y, \sigma) * f(x, y)$$

Then, the result of applying the Laplacian operator:

$$\nabla^2 L = L_{xx} + L_{yy}$$

is computed, which usually results in strong positive responses for dark blobs of radius $r = \sqrt{2\sigma}$ and strong negative responses for bright blobs of similar size.

Laplacian of the Gaussian (LoG)

Remarks

Calculating Laplacian is computationally expensive.

The Laplacian of the Gaussian operator $\nabla^2 L(x, y, \sigma)$ can also be computed as the limit case of the difference between two Gaussian smoothed images (DoG):

$$\nabla^2 L(x, y; \sigma) \approx \frac{\sigma}{\Delta\sigma} \big(L(x, y; \sigma + \Delta\sigma) - L(x, y; \sigma) \big).$$

Laplacian of the Gaussian (LoG)

Remarks

A main problem when applying LoG operator at a single scale, however, is that the operator response is strongly dependent on the relationship between the size of the blob structures in the image domain and the size of the Gaussian kernel used for pre-smoothing. In order to automatically capture blobs of different (unknown) size in the image domain, a multi-scale approach is therefore necessary.

Histogram of oriented gradients (HOG)

Histogram of oriented gradients (HOG)

Step 1— preprocess the data

Often underestimated, it is an essential part of any algorithm. Preprocessing data is a crucial step in any machine learning project and that’s no different when working with images.

We need to preprocess the image and bring down the width to height ratio to 1:2. Because, to extract the features, we will be dividing the image into 16 x 16 and 8 x 8 blocks, each dimension should be a multiple of 16. In the original paper the size 64 x 128 was used, but this is not strictly required. Rectangular shape was a consequence of the primary aim: this type of detection was used in the task of pedestrian recognition.

Histogram of oriented gradients (HOG)

Step 2— calculate gradients

Calculate gradient. You can use any formula you want, but in most cases we calculate gradients as:

$$M_x(x, y) = I(x + 1, y) - I(x - 1, y)$$

$$M_y(x, y) = I(x, y + 1) - I(x, y - 1)$$

Histogram of oriented gradients (HOG)

Step 3 — calculate the magnitude and orientation

Having gradients, you can calculate gradient direction:

$$e_{dir}(x, y) = \tan^{-1}\left(\frac{M_y(x, y)}{M_x(x, y)}\right)$$

and its magnitude:

$$e_m(x, y) = \sqrt{M_x(x, y)^2 + M_y(x, y)^2}.$$

Histogram of oriented gradients (HOG)

Step 4 — create histograms using gradients and orientation

Method 1

Create 360 slots (bins, buckets) dividing equally full circle into 1 degree parts.

For every pixel, get it angle and add 1 to the slot corresponding to this angle.

In result you obtain frequency table which can be used to generate a histogram with angle values on the x-axis and the frequency on the y-axis.

Histogram of oriented gradients (HOG)

Step 4 — create histograms using gradients and orientation

Method 2

This method is similar to the previous method, except that now you divide full circle into, for example, 10-degree slots. So, the number of buckets you would get for this case is 36.

Histogram of oriented gradients (HOG)

Step 4 — create histograms using gradients and orientation

Method 3

In this method, instead of using the frequency (instead od adding 1), you can use the gradient magnitude to fill the values in the table.

Histogram of oriented gradients (HOG)

Step 4 — create histograms using gradients and orientation

Method 4

Each pixel broadcasts a weighted vote to other slots, based on the values found in the gradient computation.

For example, if you have 36 slots, 10 degree each, and for one pixel you obtain direction 17 and magnitude 12 then you should add something to the slot 0-9, 10-19 and 20-29. Amount you add should correspond to relation between slot range and orientation — the higher contribution should be to the bin value which is closer to the orientation. For example you can compute this as follow:

$$d_1 = \left| 0 + \text{floor}\left(\frac{9-0}{2}\right) - \text{angle} \right| = |4 - 17| = 13$$

$$d_2 = \left| 10 + \text{floor}\left(\frac{19-10}{2}\right) - \text{angle} \right| = |14 - 17| = 3$$

$$d_3 = \left| 20 + \text{floor}\left(\frac{29-20}{2}\right) - \text{angle} \right| = |24 - 17| = 7$$

$$d = d_1 + d_2 + d_3 = 23$$

$$\Delta_{\text{slot 0-9}} = \frac{1}{2}\left(1 - \frac{d_1}{d}\right) \text{magnitude} = \frac{1}{2}\left(1 - \frac{13}{23}\right) 12 = 0.43 \cdot 6 = 2.6$$

$$\Delta_{\text{slot 10-19}} = \frac{1}{2}\left(1 - \frac{d_2}{d}\right) \text{magnitude} = \frac{1}{2}\left(1 - \frac{3}{23}\right) 12 = 0.86 \cdot 6 = 5.21$$

$$\Delta_{\text{slot 20-29}} = \frac{1}{2}\left(1 - \frac{d_3}{d}\right) \text{magnitude} = \frac{1}{2}\left(1 - \frac{7}{23}\right) 12 = 0.69 \cdot 6 = 4.17$$

Histogram of oriented gradients (HOG)

Step 4 — create histograms using gradients and orientation

The histogram of oriented gradients is not computed for the whole imag but for small blocks into which image is divided — 8 x 8 pixels is popular choice.

For each region we obtain one histogram. In consequence shape of data matrix is changed.

Example:

Image initial size (width x height) **in pixels**: 64 x 128

Small block size **in pixels**: 8 x 8

Image size **in regions**: 8 x 16

Histogram size for region: h

Size of new matrix data (matrix with histograms): 8 x 16 x h

Note: On original paper to computer histogram method similar to method 4 but a little bit different was used. Moreover, only degrees from 0 to 180 were considered, and histogram had 9 bins (20 degree each).

Histogram of oriented gradients (HOG)

Step 5— normalize histogram

The gradients of the image are very sensitive to the overall lighting. This will influence also our histograms calculated for 8 x 8 pixels blocks.

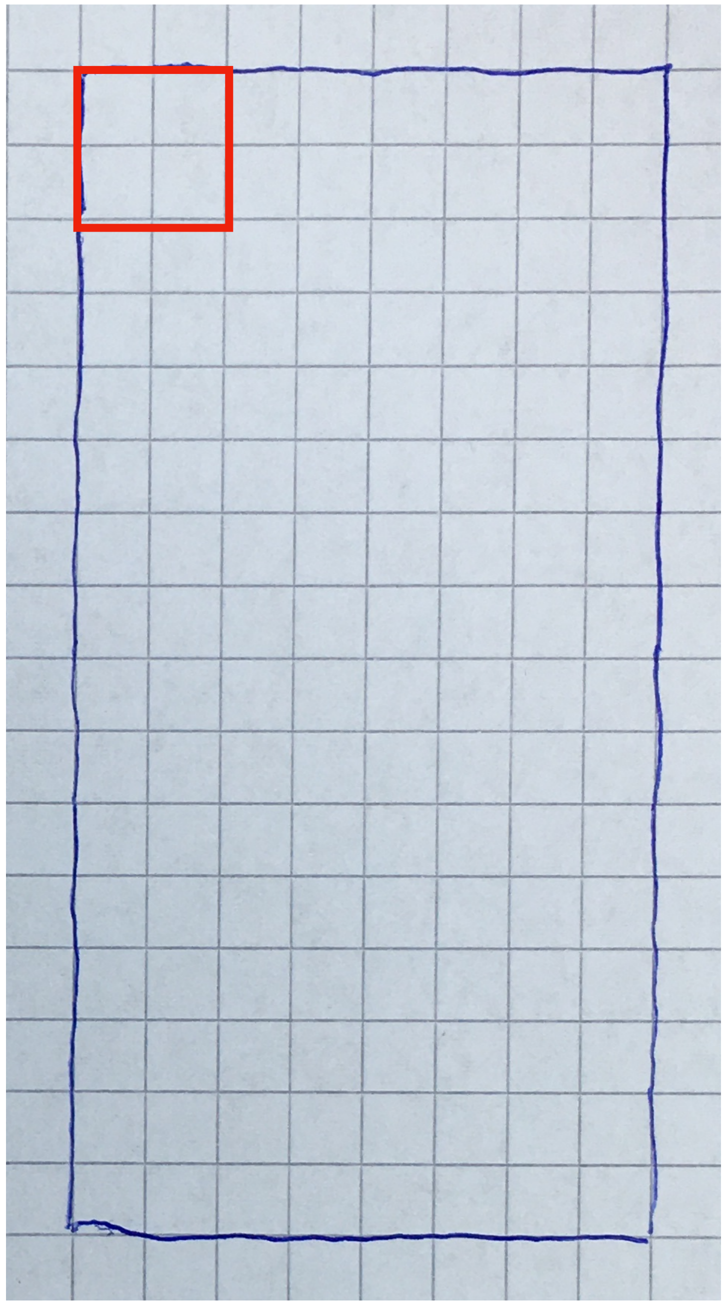
It is difficult (or even impossible) to totally get rid of this unwanted behavior, but we can reduce lighting variation by normalizing the gradients. To do this, we consider **overlapping** big blocks of the size 16×16 pixels, with a stride of 8 pixels.

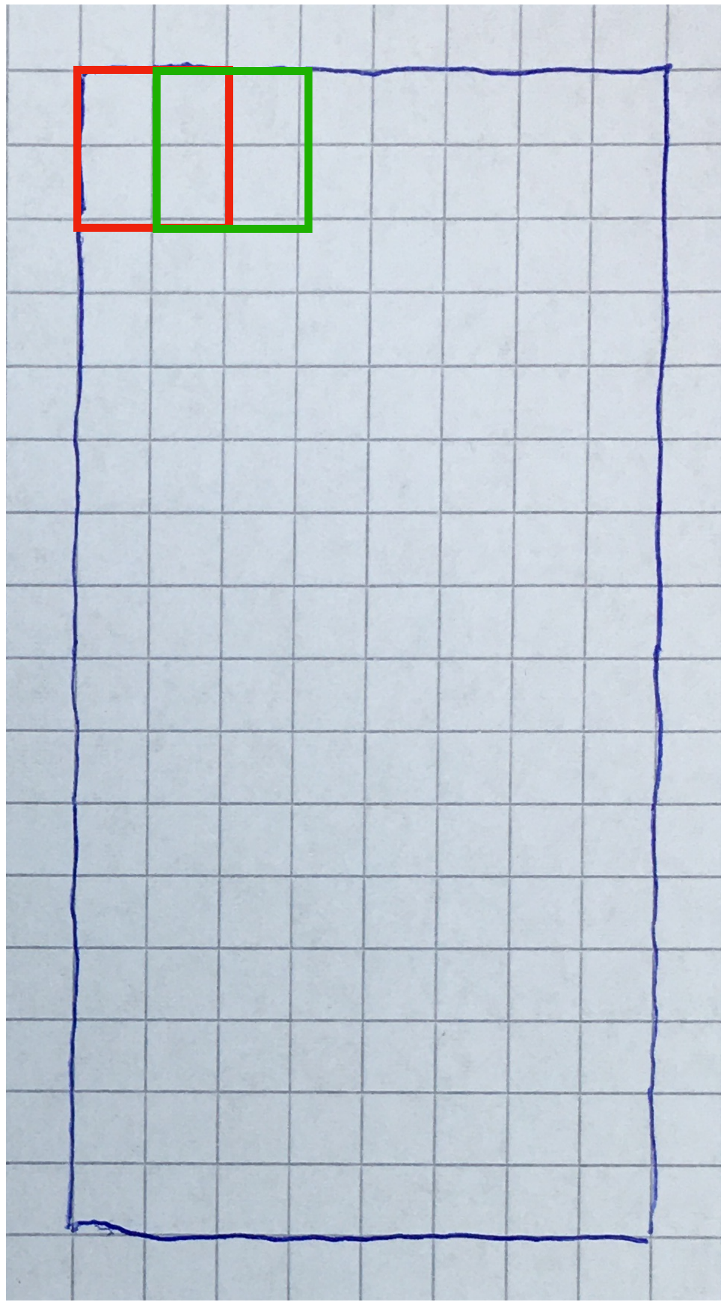
Other words, we combine **four** 8×8 blocks to create a one 16×16 big block.

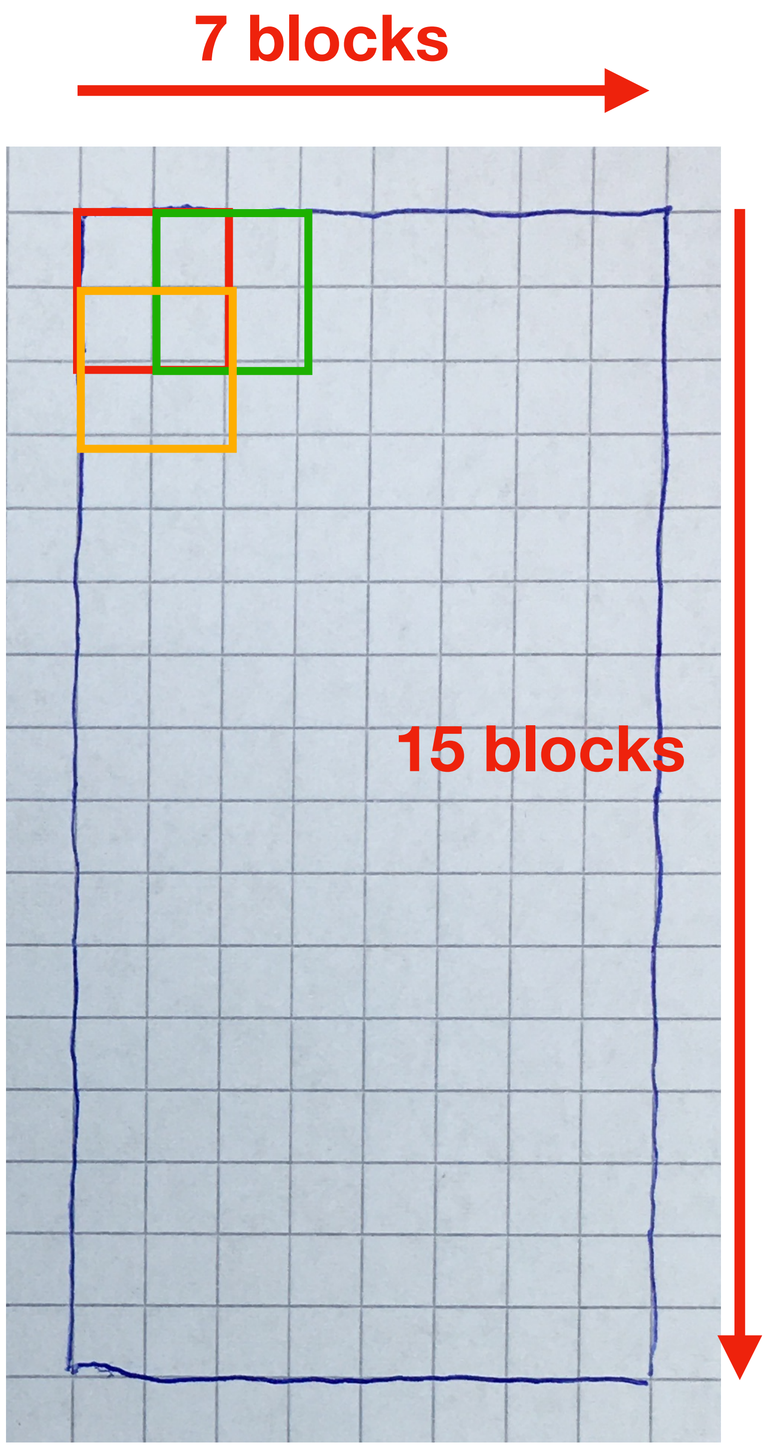
Histogram of oriented gradients (HOG)

Step 5— normalize histogram

Other words, we combine **four** 8 × 8 blocks to create a one 16 × 16 big block.


One block


Two overlapping blocks in one row



Three overlapping blocks in two rows

Histogram of oriented gradients (HOG)

Step 5— normalize histogram

Each 8 × 8 block has a $h \times 1$ matrix for a histogram. So, we would have four $h \times 1$ matrices or a single $4h \times 1$ matrix for one 16 x 16 pixels big block.

To normalize this matrix, we will divide each of these values by the square root of the sum of squares of the values.

Histogram of oriented gradients (HOG)

Step 6— features for the complete image

From the last step we have features for each 16 × 16 blocks of the image — in this case this is a normalized $4h$ × 1 matrix combining 4 histograms from 8 x 8 pixels blocks.

Now our task is to combine all of them to get the features for a whole image. We use a very simple approach — all $4h$ × 1 matrices are combined to give one, one-dimensional vector of the size:

$(4h \times 1) \cdot$ number_of_big_blocks.

Histogram of oriented gradients (HOG)

Step 6— features for the complete image

$(4h \times 1) \cdot$ number_of_big_blocks

How many big blocks do we have? The number of big blocks for one dimension is equal to:

$$\frac{\text{dimension\_in\_pixels}}{8} - 1$$

In case of 64 x 128 pixels image we have:

$$\left(\frac{64}{8} - 1\right)\left(\frac{128}{8} - 1\right) = (8-1)(16-1) = 7 \cdot 15 = 105$$

In consequence final feature vector has length of:

$$4h \cdot 105 = 420h$$

Histogram of oriented gradients (HOG)

Python code from scratch as well as with skimage library can be found in [5].

Scale invariant feature transform (SIFT)

Scale-Invariant Feature Transform (SIFT)

SIFT stands for Scale-Invariant Feature Transform and was first presented in 2004, by D.Lowe, University of British Columbia. SIFT is invariance to image scale and rotation.

This algorithm is really effective:

Image source: [1]

Scale-Invariant Feature Transform (SIFT)

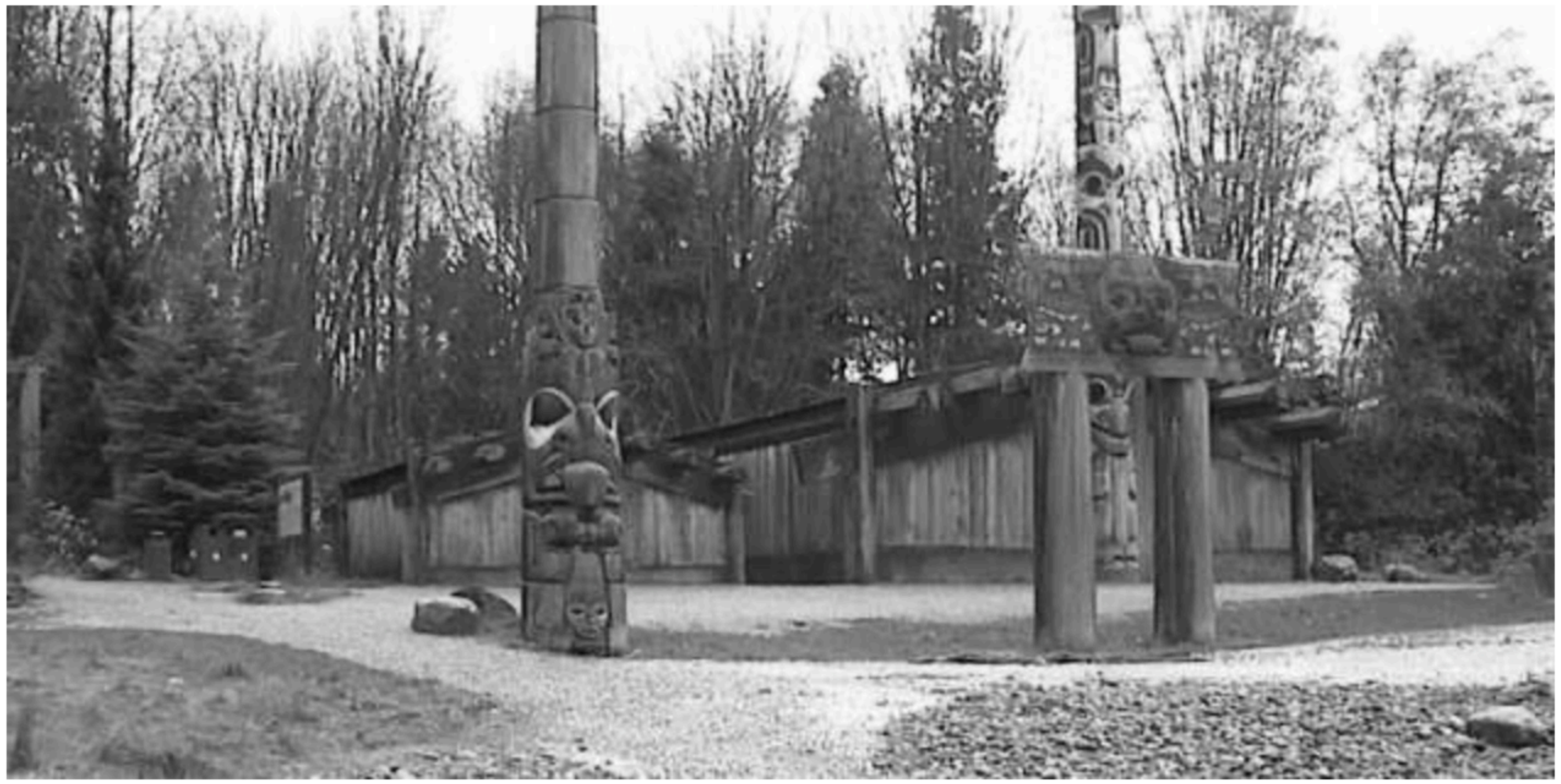


Image source: [1]

Scale-Invariant Feature Transform (SIFT)

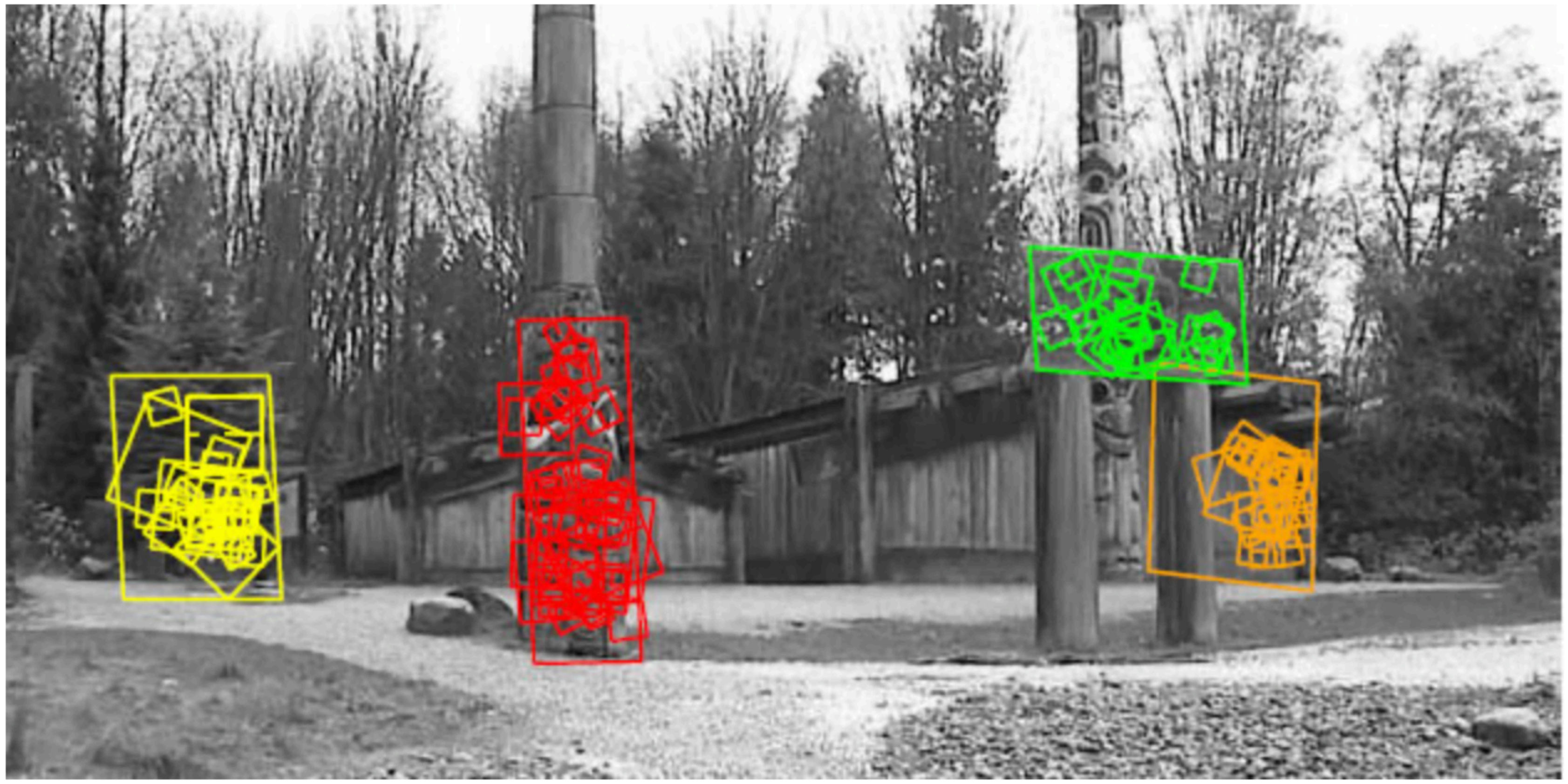


Image source: [1]

Scale-Invariant Feature Transform (SIFT)

The algorithm was patented, but expired in March of 2020.

Advantages of SIFT are

- Locality: features are local, so robust to occlusion and clutter (no prior segmentation).
- Distinctiveness: individual features can be matched to a large database of objects.
- Quantity: many features can be generated for even small objects.
- Efficiency: close to real-time performance.
- Extensibility: can easily be extended to a wide range of different feature types, with each adding robustness.

Scale-Invariant Feature Transform (SIFT)

Scale-Invariant Feature Transform (SIFT)

Step 1 — construct a scale space

This is the initial preparation. You create internal representations of the original image to ensure scale invariance. This is done by generating a *scale space*.

1. To create it, you take the original image and generate **progressively blurred out images**.

2. Resize the non-blurred image from the last step to half size and again generate blurred out images.

3. Repeat previous step few times.

In result you obtain series of images of different sizes and few different blur levels.

Scale-Invariant Feature Transform (SIFT)

Step 1 — construct a scale space

In result you obtain series of images of different sizes and few different blur levels.

Suggested number of different sizes (known as octaves) is 4 with 5 different blur levels each.

Surprisingly, to get more keypoints you can start with double original image.

Blurring can be done with many different methods, but in most cases it is referred to as the convolution of the Gaussian operator and the image.

If you decide to use Gaussian blur, then blur parameter σ in the next image should be $k\sigma$, where k is a constant you choose.

Scale-Invariant Feature Transform (SIFT)

Step 2 — calculate Difference of Gaussian

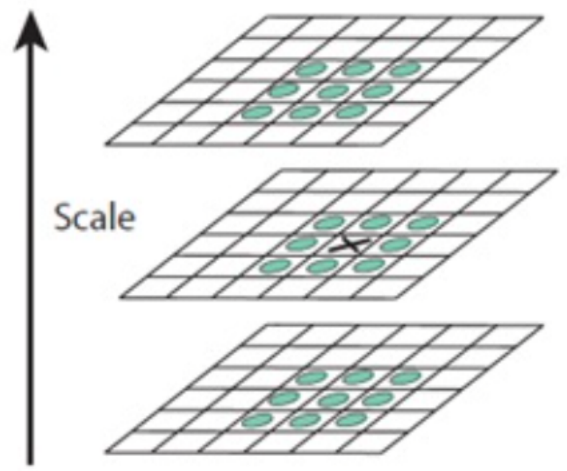
Blurred images are used to generate another set of images being results of applying Difference of Gaussians (DoG) operator.

We do this because difference of blurred Gaussians is a good approximation of the Laplacian of the Gaussian (LoG) which is stron edges and corners predictor but is also computationally expensive.

Scale-Invariant Feature Transform (SIFT)

Step 3 — find keypoints

The first step is to coarsely locate the maxima and minima. For a fixed scale you iterate through each pixel and check all it's 3D neighbours. Yes, 3D neighbours, because check is done within the current image, and also the next and previous in scale axis:



In the above image `x` marks the current pixel. The green circles mark a total of 26 neighbour fields.

`x` is marked as a candidate to be a (local) *key point* if it is the greatest or least of all 26 neighbours.

Scale-Invariant Feature Transform (SIFT)

Step 4 — Getting rid of low contrast keypoints

The number of candidate keypoints generated in the previous step may be quite big. Most of them are not useful as features — some of them lie along an edge, or they don't have enough contrast.

To get rid of useless keypoints the approach similar to the one used in the Harris Corner Detector is applied. Of course you don't have to calculate it for every image point but only for keypoints.

You also rejected candidates for keypoint if it has a low contrast.

Scale-Invariant Feature Transform (SIFT)

Step 5 — orientation assignment

At this stage, we have a set of stable keypoints for the images. We will now assign an orientation to each of these keypoints so that they are invariant to rotation. We can again divide this step into two smaller steps:

1. Calculate the magnitude and orientation.

2. Create a histogram for magnitude and orientation.

Scale-Invariant Feature Transform (SIFT)

Step 5.1 — orientation assignment (orientation and magnitude)

For each keypoint location, depending on the scale, an orientation collection region is considered. The bigger the scale, the bigger the collection region (the window size is equal to the size of the kernel for Gaussian Blur of amount 1.5σ).

For every pixel from this region magnitude and orientation is calculated as we do many times before:

$$e_m(x,y) = \sqrt{M_x(x,y)^2 + M_y(x,y)^2},$$

$$e_{dir}(x,y) = \tan^{-1}\left(\frac{M_y(x,y)}{M_x(x,y)}\right)$$

In the simplest case:

$$M_x(x,y) = I(x+1,y) - I(x-1,y)$$

$$M_y(x,y) = I(x,y+1) - I(x,y-1)$$

Scale-Invariant Feature Transform (SIFT)

Step 5.2 — orientation assignment (histogram)

To make a histogram a whole range of $360°$ is divided into 36 slots (bins), each 10 degrees.

For all the pixels around the keypoint we take corresponding angle and magnitude. Angle determines bin and magnitude is added to the bin value.

When completed, the histogram will have a peak for some slot. Angle of this slot is assigned as an orientation of this keypoint.

Also, any peaks above 80% of the highest peak are converted into a new keypoint. This new keypoint has the same location and scale as the original, but its orientation is equal to the other peak.

Scale-Invariant Feature Transform (SIFT)

Step 6 — keypoint descriptor

At the end of step 5, each keypoint has a location, scale, orientation.

Now the task is to compute a descriptor for the local image region about each keypoint that is highly distinctive and invariant as possible to variations such as changes in viewpoint and illumination.

Scale-Invariant Feature Transform (SIFT)

Step 6 — keypoint descriptor

1. Take a 16×16 neighborhood around the keypoint.
2. The 16×16 block further divided into 16 sub-blocks 4×4.
3. For each sub-blocks generate the histogram using magnitude and orientation. In this case full circle is divided into 8 bins, 45 degrees each.

Because we have 16 sub-block and in each we have one 8-bin histogram, so in total we have 16 x 8 bin values.

Potentially this 128 element vector could be used as a keypoint descriptor. It would be possible after some tuning.

Scale-Invariant Feature Transform (SIFT)

Step 6 — keypoint descriptor

- Remove rotation dependence
  The feature vector uses gradient orientations, so if we rotate the image, all orientations in keypoint descriptor change.
  To achieve **rotation independence**, the keypoint's rotation is subtracted from each orientation. Thus each gradient orientation is relative to the keypoint's orientation.

- Remove illumination dependence

  - Normalized the vector to unit length in order to enhance invariance to affine changes in illumination.

  - Apply a threshold of 0.2 (any value greater than 0.2 is changed to 0.2) and again normalize the vector.

In Euclidean geometry, an *affine transformation*, or an affinity (from the Latin, affinis, "connected with"), is a geometric transformation that preserves lines and parallelism but not necessarily distances and angles.

Scale-Invariant Feature Transform (SIFT)

Feature matching

Feature matching is done with RANSAC Algorithm.

See [3, 4] for an example with matching features and general SIFT usage in OpenCV.

Speeded up robust features (SURF) — faster version of SIFT

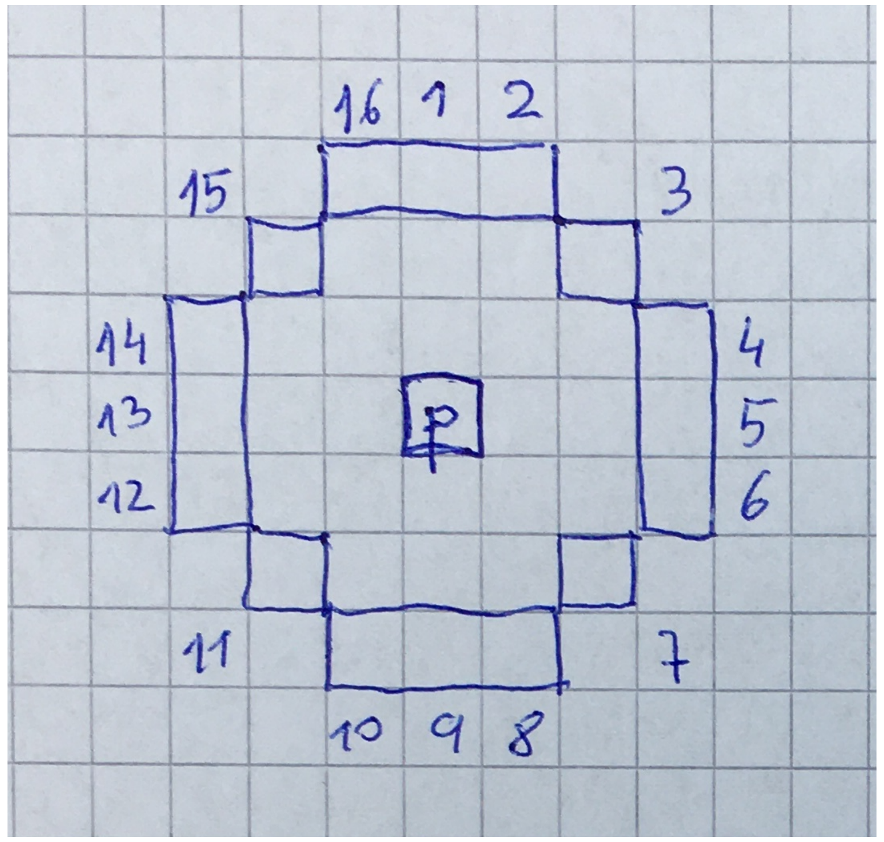
(BRIEF)

Features from accelerated segment test (FAST)

Features from accelerated segment test (FAST)

Step 1 — preliminary step

1. Select pixel p of coordinates (x_p, y_p).
2. Denote intensity of pixel p as I_p (or $I(x_p, y_p)$).
3. Choose threshold value t.
4. Consider a circle C_p of 16 pixels around the pixel p (technically this is a Bresenham circle of radius 3):



Features from accelerated segment test (FAST)

Step 2 — check corner conditions

1. The pixel p is a corner if there exists a set of n contiguous pixels in the circle (of 16 pixels) which are all brighter than $I_p + t$, or all darker than $I_p - t$.

Notes:

1. Note: In the first version of the algorithm the authors have used $n = 12$.

2. To make the algorithm fast, first compare the intensity of pixels 1, 5, 9 and 13 of the circle with I_p. From the circle figure we infer that at least three of these four pixels should satisfy the threshold criterion. If it is true then check for all 16 pixels and check if 12 contiguous pixels fall in the criterion. If not, reject the pixel p as a possible interest (corner) point.

Features from accelerated segment test (FAST)

Speed improvement with machine learning

1. Select a set of images for training (preferably from the target application domain).

2. Run FAST algorithm in every image to find feature points.

3. For every feature point p, store the 16 pixels around it as a vector f_p. Do it for all the images to get a set of feature vectors.

4. Each pixel (name it x) in these 16 pixels can have one of the following three states:

$$s_{p,x} = \begin{cases} d & I_{p,x} \leq I_p - t \quad \text{(darker)} \\ s & I_p - t \leq I_{p,x} \leq I_p + t \quad \text{(similar)} \\ b & I_p + t \leq I_{p,x} \quad \text{(brighter)} \end{cases}$$

5. For every feature vector f_p and every x, replace x feature value of this vector with corresponding $s_{p,x}$; denote resulting vector as $\bar{f}_p$.

Features from accelerated segment test (FAST)

Speed improvement with machine learning

6. Define a variable t_p which is *true* if p is an interest point (corner) and *false* if p is not an interest point.

7. At this moment you have a learning set L of pairs $(\bar{f}_p, t_p)$. Apply to this set any machine learning method you want. One of the simplest is ID3 used to generate a decision tree.

Features from accelerated segment test (FAST)

Eliminating insignificant interest points

There are many method of completing this process. One of the simplest is given below:

1. For each of interest point p compute a score value v_p as the sum of the absolute difference between the pixels intensity from a circle $I_{p,x}$ and the intensity I_p of the point p.

2. Consider two "adjacent" interest points p and q, compare their score values v_p and v_q. Discard the one with the lower score value.

Notes:

1. There may be different definitions of adjacency.

2. Discarding procedure may be very simple or much more sophisticated. We can imagine situation with three points p, q and r where q is adjacent to p, r is adjacent to q and $v_p > v_q > v_r$. In this case order of discarding may affect final result.

Oriented FAST and Rotated BRIEF (ORB)

Locally Contrasting Keypoints (LOCKY)

Bibliography

Bibliography

1. David G. Lowe, Distinctive Image Features from Scale-Invariant Keypoints,
   https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf
   January 5, 2004,
   retrieved, 2021-11-21

2. Scale space,
   https://en.wikipedia.org/wiki/Scale_space
   retrieved, 2021-11-21

3. Aishwarya Singh, A Detailed Guide to the Powerful SIFT Technique for Image Matching (with Python code),
   https://www.analyticsvidhya.com/blog/2019/10/detailed-guide-powerful-sift-technique-image-matching-python/
   October 9, 2019,
   retrieved, 2021-11-21

4. Deepanshu Tyagi, Introduction to SIFT(Scale Invariant Feature Transform),
   https://medium.com/data-breach/introduction-to-sift-scale-invariant-feature-transform-65d7f3a72d40
   Mar 16, 2019,
   retrieved, 2021-11-21

5. Mrinal Tyagi, HOG (Histogram of Oriented Gradients): An Overview,
   https://towardsdatascience.com/hog-histogram-of-oriented-gradients-67ecd887675f
   retrieved, 2021-11-21